ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

IMPLEMENTAZIONE DEL BACK-END DI UNA APP ANDROID PER LA DIDATTICA DELLA DEDUZIONE NATURALE

Relatore:
Chiar.mo Prof.
CLAUDIO SACERDOTI COEN

Presentata da:
DANILO BERARDINELLI

Sessione II

A.A. 2013/2014

Indice

Capitolo 1:
Premessa

Questa tesi si occupa della progettazione e dello sviluppo di un'app didattica per la visualizzazione e la risoluzione di alberi di Deduzione Naturale.
L'applicazione nasce dall'idea di fornire un nuovo approccio all'attività di laboratorio del corso di Logica per l'Informatica, nell'ambito del quale viene attualmente usato il theorem prover "Matita".
L'utilizzo di tale software si è infatti rivelato in parte problematico per gli studenti, principalmente a causa della complessa interfaccia, difficile da padroneggiare per un utente alle prime armi.
L'idea dell'applicazione realizzata è di offrire un'interfaccia più intuitiva, basata su touch screen e più vicina alla realizzazione di alberi "su carta" rispetto a quella offerta da Matita.
Nell'ambito del progetto viene inoltre offerta una piattaforma per la gestione di esercizi: il docente ha a disposizione una pagina Web da cui uplodare esercizi di deduzione naturale in formato XML che lo studente può scaricare e svolgere utilizzando il proprio dispositivo Android.
L'applicazione si occuperà anche di valutare gli utenti e di registrare il punteggio di ogni esercizio con relativo voto su un database, che può essere controllato e gestito dal docente del corso.

Capitolo 2:
Interfaccia Utente

L'applicazione prevede due diverse tipologie di utente: il docente (o l'amministratore) e gli studenti.
Agli studenti viene fornita un'applicazione per dispositivi Android su cui è possibile risolvere, previa l'autenticazione con il server, gli esercizi messi a disposizione dal docente. Visto che l'applicazione è di tipo didattico ed è pensata per gli studenti di Bologna, è possibile registrarsi solo se si è in possesso di un email "@studio.unibo.it". È da notare che occorre registrarsi e che non è possibile accedere direttamente con le credenziali di ateneo.
È anche possibile per l'utente creare i propri esercizi utilizzando la sintassi XML dell'applicazione: tali esercizi non saranno tuttavia valutati, né registrati nel database del docente.
Una volta che uno studente effettua l'autenticazione, è possibile lavorare off-line, effettuando l'aggiornamento quando lo si ritiene necessario. Si è scelta questa soluzione per permettere una maggiore durata della batteria, in modo da far pesare il meno possibile l'utilizzo dell'applicazione. Alla fine di ogni esercizio viene assegnato un voto da 1 a 5; una volta effettuata la sincronizzazione, il voto viene memorizzato nel database del server. Nella schermata degli esercizi verrà visualizzato il voto migliore di un esercizio di fianco al suo nome.
Per l'amministratore è stata creata una pagina Web attraverso la quale si possono caricare/eliminare esercizi o visualizzare/eliminare informazioni riguardanti gli studenti. Il docente si occupa anche di scrivere gli esercizi e testare che questi siano risolvibili; per ogni esercizio, inoltre, deve fornire delle informazioni riguardanti la soluzione ottimale

in modo che l'applicazione possa valutare gli studenti in base a queste informazioni.

La correttezza sintattica degli esercizi viene controllata dal server centrale che, nel caso in cui non ci siano problemi, si preoccupa di inviarli e di farli visualizzare agli studenti.

Capitolo 3:
Implementazione

L'intero programma è stato diviso in due parti sviluppate autonomamente e separatamente, che possiamo identificare come “backend” e “frontend”.
La parte di mia competenza (il backend) ha riguardato la memorizzazione dei dati e la gestione delle comunicazioni tra gli utenti e il server centrale.
Inoltre, ho gestito l’interfaccia di comunicazione tra backend e frontend attraverso un parser XML (che si occupa di convertire gli esercizi in XML del server nella struttura dati utilizzata dal frontend) e ho implementato un editor interattivo basato sui Dialog di Android che dà all’utente la possibilità di inserire nuove formule nell’ambito della dimostrazione.
Infine, mi sono occupato di gestire le metriche di valutazione degli esercizi e di stabilire dei parametri oggettivi per giudicare la “bontà” di una soluzione fornita.
Facendo una panoramica del backend possiamo dividerlo in quattro parti:

-Il server: gestisce la comunicazione con l'applicazione e si occupa di gestire i dati degli utenti e degli esercizi;

-L'applicazione: si occupa di recuperare gli esercizi messi on line dal docente e di inviare al server le informazioni relative all'utente e agli esercizi che egli ha svolto;

-La Web app: è lo strumento di amministrazione del docente, mostra le informazioni raccolte dal server e permette il caricamento degli esercizi;

-I database: due per la precisione, il primo usato dall'applicazione il dal secondo server . Si occupano di memorizzare i vari dati utili alle due parti.

Il server e la Web app sono due entità separate. Infatti la Web app si occupa solo di gestire l'interazione con l'amministratore mentre il server si occupa delle informazioni sugli utenti e gli esercizi: Ho preferito quindi dividere le due parti, utilizzando per ognuna di esse, il linguaggio più consolo all'uso che dovevo farne (Java per il server e PHP per la Web app). Prima di iniziare a parlare delle varie parti è bene aprire una parentesi sul protocollo di comunicazione utilizzato e sulla sintassi XML riguardante gli esercizi.

3.1 Protocollo di comunicazione:

Per quanto riguarda la comunicazione ho scelto di implementare un protocollo ad hoc. Questa scelta è dovuta alla semplicità di implementazione richiesta: infatti il protocollo è abbastanza basilare e si prestava bene ad un implementazione attraverso stringhe che ne ha mantenuto la leggibilità e la semplicità implementativa.
Di seguito, quando parleremo delle varie parti del progetto, verranno elencati tutti i tipi di richiesta da parte dell'applicazione e le relative risposte del server.
Passare comunque ad uno standard non risulta essere complicato. Infatti le informazioni vengono parsate da una classe, chiamata "suString", che si occupa di prendere le informazioni, trasformarle in stringhe riconosciute

dal protocollo e viceversa. Per cambiare protocollo quindi basta sostanzialmente modificare la suddetta classe.

3.2 Sintassi XML degli esercizi:

Per quanto riguarda gli esercizi, è stato scelto di utilizzare un grammatica ad hoc. Per gli esercizi infatti non ci sono solo le classiche informazioni che riguardano le tesi e le ipotesi, ma anche quelle riguardanti i parametri di valutazione.
Il file è diviso in tre parti: ipotesi, tesi, valutazione; scritti nel modo seguente:

```
<esercizio>
     <ipotesi>
          //formule delle ipotesi
     </ipotesi>
     <tesi>
          //formula della tesi
     </tesi>
     <valutazione>
          <click>numero di click</click>
          <tempo>tempo impiegato</tempo>
          <nodi>numero nodi</nodi>
     </valutazione>
</esercizio>
```

Le ipotesi e la tesi sono costituite da formule (una sola per quanto riguarda la tesi) che può essere di 6 tipi, specificati nel parametro "type":
-impl: indica l'implicazione e ha due formule come figli;
-and: indica l'unione e ha due formule come figli;
-or: indica la disgiunzione e ha due formule come figli;
-not: indica negazione e ha come figlio una formula

-atomic: indica che nel corpo sono presenti top (indicato con “top”) o bottom (indicato con “bot”). Questo type non ha figli;
-literal: indica che nel body è presente una variabile proposizionale (indicata con una sola lettera). Anche questo type non ha figli.
Per passare da questa grammatica di scrittura ad una “standard” (come ad esempio Nollosow) basta modificare la classe “parser” presente sia nel client che nel server.

3.3 Sviluppo dell’applicazione Android

Per quanto riguarda questa parte del progetto, mi sono occupato in un primo momento della comunicazione client/server e della sua riservatezza. Infatti tutte le comunicazioni sono cifrate, viene utilizzato un certificato autofirmato disponibile sia sul client che sul server. Inoltre si è cercato di ridurre al minimo l’utilizzo delle credenziali dell’utente.
In un secondo tempo invece mi sono occupato di effettuare il parsing degli esercizi e delle formule inserite dall’utente durante lo svolgimento delle prove. Questo paragrafo verrà quindi suddiviso nelle seguenti quattro parti:

-Meccanismi per la gestione delle credenziali;
-sincronizzazione delle informazioni da client a server e viceversa;
-Conversione degli esercizi dal formato XML alla struttura dati utilizzata a run-time dal frontend;
- Parsing delle formule inserite direttamente dall’utente. Più propriamente è stato implementato un editor interattivo (basato su *AlertDialog*) per consentire all’utente di scaricare nuove ipotesi nel corso della dimostrazione.

Prima di parlare delle diverse richieste che è possibile fare al server analizziamo brevemente il protocollo utilizzato nella comunicazione.
Le stringhe riconosciute dal server sono del tipo:

richiesta/campo1/campo2/campo3....

la classe suString ha i vai metodi per convertire una stringa del genere in un ArrayList o in un vettore di stringhe e viceversa.
L'intero protocollo viene sviluppato a livello transport e la comunicazione avviene tramite socket SSL. Tutte le comunicazioni, infatti, sono criptate: il certificato relativo alla comunicazione è presente sia sul server in formato JKS, sia sul dispositivo in formato BKS (l'unico formato riconosciuto da Android). Nel client la comunicazione è gestita dal metodi "connessione" o "connessioneMain" della classe serverComunication.

3.3.1 Gestione delle credenziali degli utenti:

In questa sezione verranno descritti tutti gli strumenti messi a disposizione allo studente per poter gestire le sue credenziali.
Il paragrafo verra diviso in quattro parti:

-registrazione;
-login;
-recupero password;
-modifica password.

dopo la descrizione di ognuna di queste parti, verrà mostrato il tipo di richiesta da effettuare relativa al protocollo utilizzato.

-Richiesta 0, registrazione:

Come ho già scritto in precedenza, per poter utilizzare l'applicazione è necessario effuttuare prima la registrazione. La view relativa a questa richiesta infatti è la prima che appare all'utente. La schermata inoltre è utilizzata per effettuare sia la registrazione che il login.

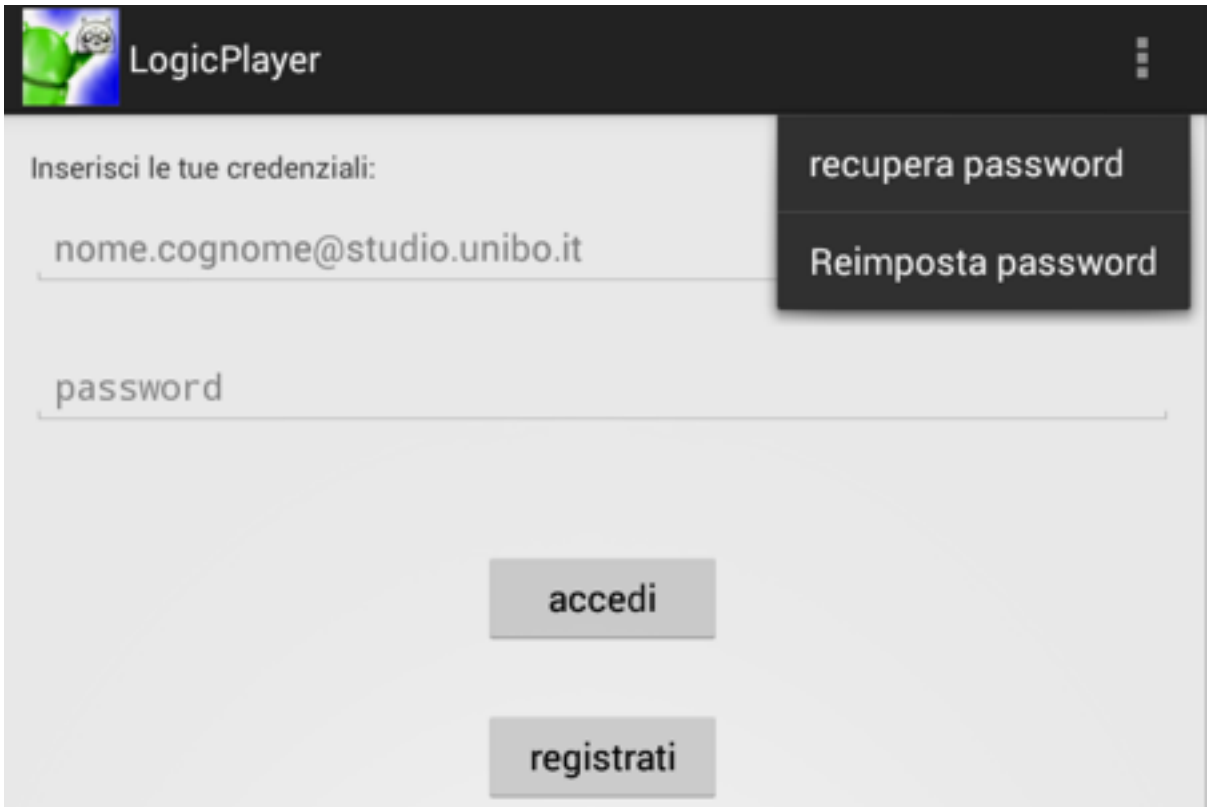

Figura 3.1

Prima che le informazioni vengono inviate al server, al momento della pressione del tasto "registrati", viene controllato che tutti i campi siano riempiti e che non contengano il carattere '/' dato che è riservato alla comunicazione.
I controlli sulle credenziali vengono effettuati direttamente dal server e prevedono, oltre all'obbligo di fornire un indirizzo '@studio.unibo.it", di inserire una password più lunga di 6 caratteri.

Richiesta: 0/username/password

-Richiesta 1, login:

Questo tipo di richiesta, può essere fatta in due momenti. Il primo è premendo il relativo tasto nella view mostrata in *Figura 3.1*, in questo caso, viene controllato se i campi non sono vuoti e non contengano '/'. Il server dopo aver controllato che le informazioni inviate hanno una corrispondenza nella tabella relativa agli utenti registrati, inserisce l'username nella tabella degli utenti connessi e invia al client un numero casuale che corrisponde alla chiave di sessione. Ogni volta voltacheche viene effettuata una connessione, vengono eliminati tutti gli utenti che non comunicano con il server per più di 10 minuti. Alla fine della comunicazione il timer viene resettato all'ora attuale.
Il secondo caso in cui viene effettuata una richiesta di login, è quando l'applicazione ha già memorizzato un utente e una password. In questa seconda opzione viene lanciata l'activity "aggiornamento" a cui vengono passate le credenziali memorizzate. L'activity si occupa, oltre che a effettuare le richieste relative per la sincronizzazione del database, di inviare la richiesta di login nel caso in cui l'utente non abbia già effettuato il login.

Richiesta: 1/username/password

-Richiesta 2, recupero della password:

Come mostrato in figura *3.1*, una delle due opzioni del menù è "Recupera password". Dopo aver inserito il proprio indirizzo email, cliccando su questa opzione, viene inviata la richiesta al server. Quest'ultimo controlla se l'utente è registrato. Nel caso in cui ci sia una corrispondenza, invia all'indirizzo inserito la password dell'utente.

-Richiesta: 2/username

-Richiesta a0, modifica password:

Cliccando la seconda opzione del menù mostrato nella figura *3.1*, si lancia l'activity che si occupa di gestire la modifica della password.

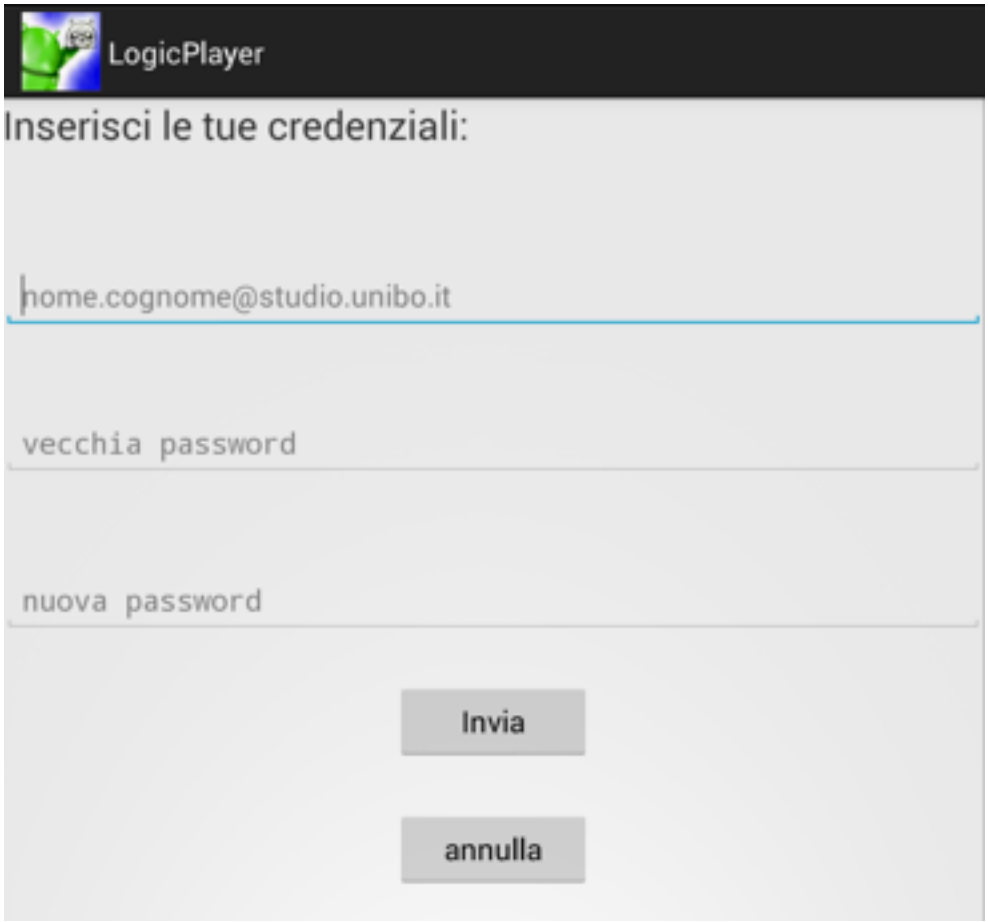

Figura 3.2

Si apre quindi una nuova schermata (mostrata in *Figura 3.2*). Da questa view è possibile, compilando il form, inviare al server la richiesta di cambio password. Anche in questo caso viene controllato che tutti i campi non siano vuoti e non contengano '/'.

Al completamento della richiesta, o nel caso in cui viene premuto il tasto "annulla", verra rilanciata la "mainActivity".

Richiesta: a0/username/vecchia password/nuova password

3.3.2 Sincronizzazione delle informazioni da client/server e viceversa:

Dopo il completamento della procedura di login l'applicazione tenta di sincronizzare tutte le informazioni presenti sul dispositivo con quelle presenti nel server. La sincronizzazione avviene su tutti i dispositivi utilizzati dall'utente che quindi non è costretto ad utilizzarne uno solo per risolvere gli esercizi.
La sincronizzazione automatica può avvenire in due punti:

-una volta effettuato il login;
-una volta terminato un esercizio.

In entrambi i casi infatti viene chiamata l'activity "aggiornamento" che una volta che termina la sincronizzazione chiama a sua volta la "download_page", l'activity che mostra tutti gli esercizi presenti sul dispositivo.
È giusto sottolineare che la sincronizzazione non preclude l'utilizzo dell'app, infatti mentre per il login l'utente è obbligato ad avere conferma da parte del server, una volta effettuato l'accesso possiamo utilizzare l'applicazione senza doverla sincronizzare.
Entrando nello specifico, la sincronizzazione con il server è divisa in tre parti:

-verifica degli esercizi: il client aggiorna il server con i nuovi esercizi svolti;
- download degli esercizi dell'utente presenti sul server: il server invia al client tutti gli esercizi che un utente ha svolto;
-download degli esercizi: scarica gli esercizi aggiornati o non presenti sul dispositivo.

Ognuna di queste parti viene gestita da un metodo contenuto nella classe "aggiorna". Come ho già scritto nel paragrafo *3.3.1,* questa classe si occupa di effettuare il login nel momento in cui un utente non è più presente nella tabella connessi. Nel caso in cui le credenziali non fossero valide, viene richiamata la "mainActivity" per far effettuare il login.

- Verifica degli esercizi

In questa prima fase il client manda al server le informazioni riguardanti gli esercizi dell'utente che non sono ancora stati validati dal server. Al termine di un esercizio infatti nel database interno del dispositivo vengono memorizzate le informazioni riguardanti: la valutazione della prova effettuata, l'MD5 del file su cui l'utente ha lavorato, il timestamp unix al momento in cui viene completato l'esercizio, il check dell'esercizio (che al momento dell'inserimento è 0), il nome del file e anche il nome dell'utente dato che un dispositivo può avere utenti differenti.
Questa operazione quindi invia al server le informazioni degli esercizi svolti dall'utente e che hanno ancora check 0. Se viene convalidato allora il relativo check verrà settato a 1 (il server a sua volta memorizza al suo interno i dati ricevuti), altrimenti le informazioni verranno cancellate dal database interno del dispositivo.
Bisogna far notare due cose: la prima è che gli esercizi creati dall'utente non verranno mai salvati nel database del server e, dopo questa fase, verranno anche eliminate da quello interno del dispositivo. La seconda è che per come è stato sviluppato il protocollo, per ogni esercizio con check 0, verra effettuata una richiesta dal client. N esercizi quindi richiederanno N richieste al server.

Richiesta: 8/sessionKey /Esercizio/MD5/time/voto

-Scaricamento degli esercizi dell'utente presenti sul server

Dopo aver mandato al server i nuovi esercizi svolti, l'applicazione effettua una richiesta perottenere tutti gli esercizi che un utente ha svolto. In questo modo tutti i dispositivi di un utente avranno le stesse informazioni. Per controllare se un esercizio è già stato svolto si fa riferimento, oltre che al nome, al momento in cui è stato terminato l'esercizio.

Richiesta: 9/sessionkey

-Download degli esercizi

Il metodo si occupa di far scaricare i nuovi esercizi presenti, e nel caso in cui qualche esercizio fosse stato aggiornato, elimina quello presente sul dispositivo e lo sostituisce con quello nuovo.
Per prima cosa il client effettua al server la richiesta per avere la lista degli esercizi presenti sulla macchina. Una volta ottenuta la lista, il client controlla se la cartella "tesiEs" contiene già un esercizio con quel nome e nel caso in cui fosse presente, se l'MD5 inviato dal server corrisponde con quello del file presente nel dispositivo.
Nal caso in cui il primo controllo non vada a buon fine, il client effettua subito lo scaricamento del file, nel secondo caso invece, bisogna prima eliminare il file dalla cartella e poi sostituirlo con la versione presente nel database.
Il client effettua il download con una richiesta di tipo HTTP all'indirizzo della Web app.
Conoscendo l'indirizzo del server in cui sono salvati gli esercizi, è possibile effettuare la richiesta anche da browser.

Richiesta: 4/sessionKey

3.3.3 Conversione dagli esercizi da xml ai dati utilizzati dall'applicazione:

Una volta terminata la sincronizzazione viene chiamata l'activity "download_page" che mostra tutti gli esercizi presenti nella cartella tesiEs:

Figura 3.3

Se l'esercizio è stato risolto mostra il voto migliore che è stato ottenuto (un utente può eseguire più volte un esercizio), altrimenti viene notificata all'utente la mancata risoluzione.

Cliccando su "Esci" è possibile tornare alla schermata di login. Questa operazione non elimina l'utente che è stato memorizzato, quindi chiudendo e riaprendo l'applicazione il login viene di nuovo effettuato automaticamente.

Cliccando su “aggiorna”, viene richiamata l’activity che si occupa dell’aggiornamento delle informazioni presenti nel database.
Una volta scelto l'esercizio che si vuole svolgere viene chiamata la DrawActivity, l’activity che si occupa di gestire la rappresentazione e la risoluzione degli alberi di Deduzione Naturale.
È da questa activity che viene chiamato il parser XML che si occupa di leggere gli esercizi e di trasformarli in un “Node”, cioè il dato che contiene tutte le informazioni che servono alla DrawActivity.
La classe “parser” contiene il metodo “root” che come dicevamo si occupa della codifica. Prima di tutto vengono recuperate le informazioni riguardanti la tesi, viene quindi chiamato un altro metodo, che ricorsivamente, visita tutto l'albero della formula e a ogni chiamata costruisce il nuovo nodo “Formula” che andrà poi a formare quella finale.
Una volta recuperate le informazioni sulla tesi vengono recuperate quelle riguardanti le ipotesi. Il procedimento è analogo con la differenza che le “Formule” che compongono l'ipotesi possono essere più di una.
Se il parser terina senza errori, restituisce un “Node” contenente tutte le informazioni recuperate. In caso di errore il parser restituisce *null* e si lascia la gestione dell'errore alla DrawActivity.

3.3.4 Parser delle formule inserite direttamente dall’utente:

La DrawActivity è l’activity che si occupa di mostrare l’albero di dimostrazione in maniera top-down.
Una volta aperto un esercizio l’applicazione mostra la formula che si intende dimostrare nella parte inferiore della schermata e in alto le ipotesi relative al nodo selezionato. Svolgendo l’esercizio, vengono aggiungi ad ogni passo della dimostrazione i nodi che compongono l’albero di deduzione.

È sempre possibile tornare indietro alla schermata degli esercizi cliccando su abbandona esercizio:

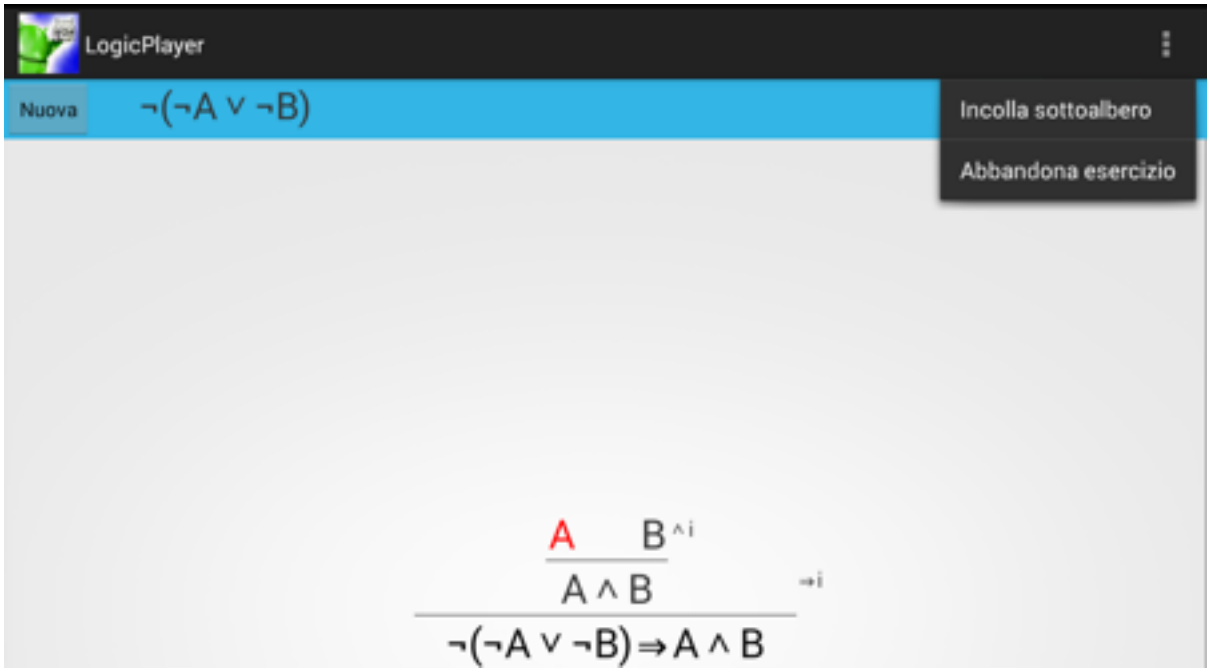

Figura 3.4

Nel corso della dimostrazione è possibile interagire con il nodo corrente in due modi distinti:

- applicando una delle *regole di introduzione* visualizzate nel Dialog a scomparsa legato al nodo attuale
- scaricando una delle ipotesi legate al nodo mediante una *regola di eliminazione*

Particolarmente interessante è il caso in cui l'utente desideri scaricare una nuova ipotesi che non sia presente nel nodo attuale, ad esempio applicando il principio del terzo escluso su uno dei letterali del teorema

In questo caso è sufficiente cliccare sull'opzione "Nuova..." situata tra le ipotesi del nodo, che fa comparire il seguente Dialog:

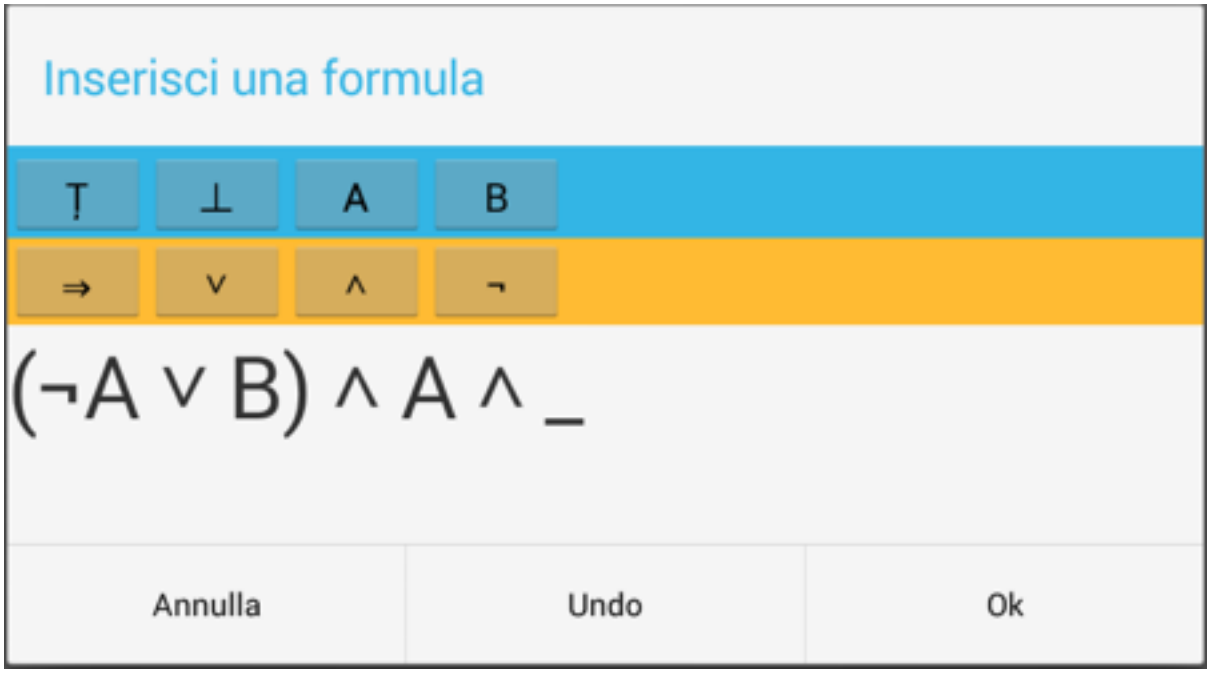

Figura 3.5

Come possiamo vedere nella view blu sono presenti tutti i letterali (più top e bottom) e in quella arancione i quattro connettivi logici.
L'inserimento della formula è orientato alla sintassi: cliccando su un connettivo questo viene inserito lasciando due buchi per le sottoformule. I successivi inserimenti andranno a riempire i buchi residui nell'ordine di visita prefissa dell'albero. È permesso inoltre inserire solo i letterali presenti nell'esercizio, che vengono ottenuti chiamando il metodo getLiteral del parser XML.
Durante la digitazione la stringa viene rappresentata utilizzando l'algoritmo di conversione in stringa e di precedenza delle parentesi già implementato all'interno della classe Formula.
Un cursore (rappresentato con '_') indica in che posizione stiamo inserendo il nuovo simbolo (che sia operando o letterale).

Il cursore viene sempre dato all'operando più a sinistra in cui possiamo inserire un simbolo, poiché per come è strutturata una TextView in Android, risultava complesso, a livello implementativo, dare la possibilità all'utente di scegliere dove inserire un simbolo.
Una volta che tutta la formula viene inserita questa viene mandata alla MyActivity per essere scaricata. Fino a quando non è stato completato l'inserimento invece non è possibile premere ok ma solo il tasto “annulla” o “undo”.
Tutto l'inserimento si basa sulla classe “Formula”: infatti grazie alla sottoclasse UndefinedFormula è possibile completare gradualmente una formula incompleta e inserirla soltanto quando è stata completata.
Il parser di stringhe è stato realizzato estendendo la classe AlertDialog di Android: come in tutti i dialog è dunque possibile in qualunque momento tornare alla MyActivity premendo il tasto “annulla”.

3.4 Sviluppo dell'applicazione Web

La Web app si presenta come una pagina Web costruita dinamicamente tramite AJAX. È molto basilare e cerca di avere tutti gli strumenti per gestire le informazioni relative agli studenti e agli esercizi. È inoltre presente il form con dal quale vengono caricati gli esercizi all’interno della macchina. Una volta caricati sul server, potranno essere scaricati dall'app.
L'applicazione Web è costituita da quattro parti da cui è possibile:

-effettuare login dell'insegnante (o amministratore);
-gestire le informazioni riguardanti gli esercizi;
-gestire informazioni riguardanti gli studenti;
-upload di file.

3.4.1 Login dell'insegnante:

Non esistono form per permettere la registrazione di un amministratore. L'unico modo per aggiungere una nuova voce è effettuare la direttamente la richiesta al database.

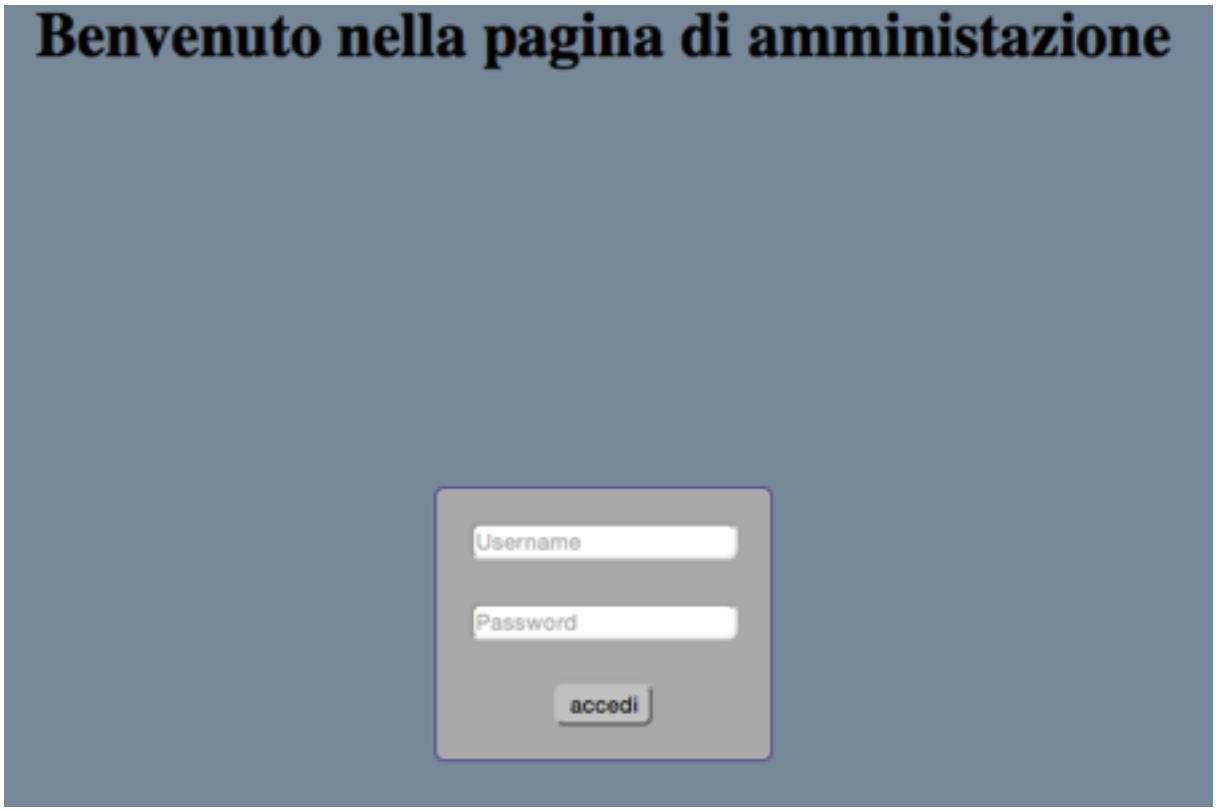

Figura 3.6

Per l'amministratore non sono previsti parametri particolari per l'username e password.

Al tasto "accedi" del form è legato un listener realizzato in JQuery.

Il listener fa partire una callback che crea dinamicamente un JSON di richiesta di login e contatta il server Apache.

Quando il server ha confermato l'avvenuto login viene settato il vettore globale $_SESSION di PHP, che implementa la sessione per l'utente.

Una volta effettuato il login compare la seguente schermata:

Figura 3.7

3.4.2 Upload dei file:

In cima alla pagina è situato il form utilizzato per l'upload: una volta scelto il file da caricare, viene lanciato lo script PHP che si occupa di gestire il caricamento file.

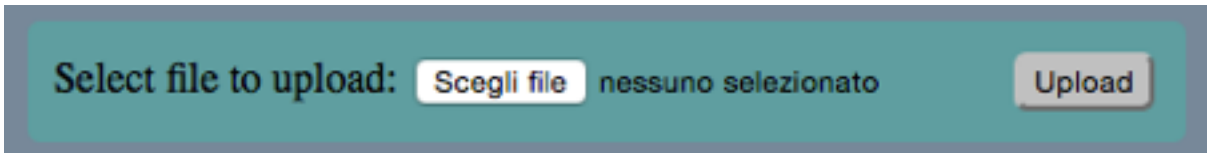

Figura 3.8

Lo script controlla se il file inserito è un file XML, lo carica sulla macchina e aspetta un secondo per controllare se il server ha accettato il file e lo ha aggiungo al catalogo degli esercizi.

Nel caso in cui il file non sia valido sarà il server stesso a preoccuparsi di eliminarlo (il procedimento verra spiegato nel dettaglio nella sezione relativa alla gestione degli esercizi *3.5.1*). In caso contrario, invece, il nome del nuovo esercizio verrà aggiunto alla lista degli esercizi disponibili.

3.4.3 Informazioni sugli esercizi:

Il catalogo degli esercizi viene visualizzato nella parte sinistra della pagina.

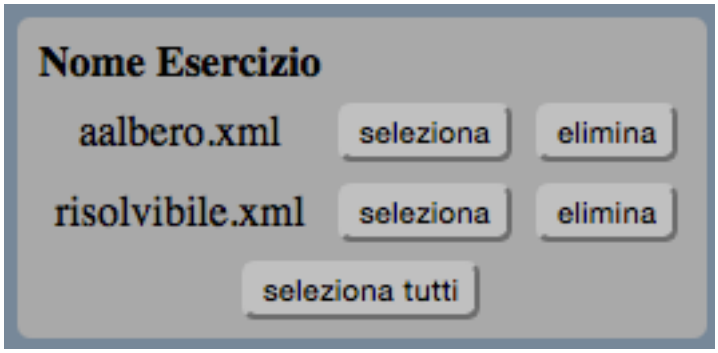

Figura 3.9

All'interno di questa sezione è possibile scegliere se avere tutta la lista degli utenti e delle valutazioni corrispondenti ad un determinato esercizio.
È possibile anche eliminare l'esercizio: in questo caso viene lanciato uno script che si occupa di eliminare il file, svegliando un thread del server che si occuperà di cancellare l'esercizio dalla lista di quelli disponibili (naturalmente verranno eliminate dal database tutte le voci relative all'esercizio).

Oltre all'opzione di visualizzazione dei risultati per il singolo esercizio, è anche possibile visualizzare i risultati dell'intero database agendo sul tasto "seleziona tutti".

Figura 3.10

A destra della pagina troviamo la tabella che mostra per ogni esercizio, chi lo ha risolto con la relativa valutazione. Premendo il tasto elimina, toglieremo questa informazione dal database.

3.4.4 Informazioni sugli utenti:

Sotto la sezione riguardante gli esercizi abbiamo quella riguardanti gli utenti: è possibile, per ogni utente, visualizzare tutti gli esercizi risolti con i relativi voti.

Figura 3.11

È anche possibile eliminare le informazioni su un utente premendo il tasto elimina, o eliminando le informazioni riguardanti un solo esercizio cliccando il tasto di fianco alla tabella degli esercizi svolti.

come possiamo vedere nella *Figura 3.12*, in questo caso per ogni studente viene segnalato l'esercizio svolto con la relativa valutazione.

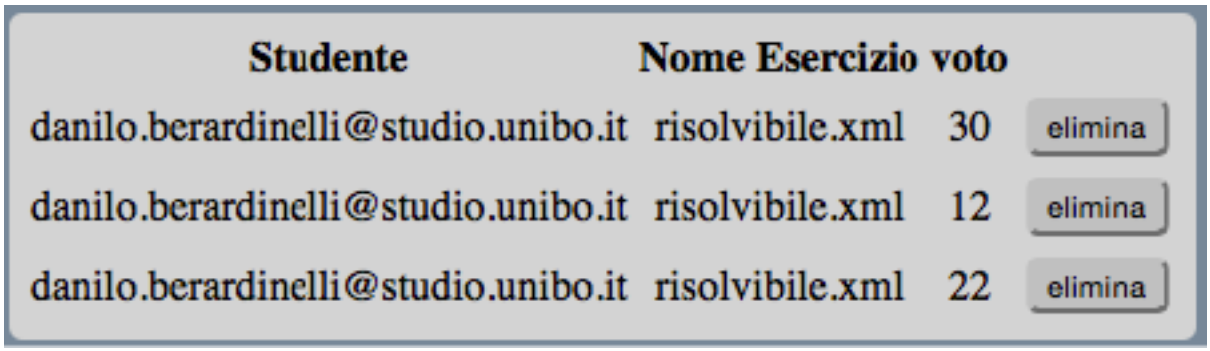

Figura 3.12

3.5 Descrizione del server

Scritto in Java 8, il server si occupa della gestione di tutti i dati, da quelli degli utenti agli esercizi inseriti. La classe principale è "superServer", che all'avvio genera due thread: uno si occuperà della gestione degli esercizi, l'altro si occuperà delle richieste HTTP per la conferma della registrazione.
Il processo centrale invece si occupa della comunicazione con il client.

3.5.1 Gestione degli esercizi:

Come abbiamo già detto nella sezione relativa all'upload dei file (*3.4.2*), è il server che si occupa di controllare la correttezza e l'aggiunta alla lista degli esercizi di un file caricato.
Il thread relativo alla gestione degli esercizi si mette in attesa sulla cartella degli esercizi. La posizione della cartella è memorizzata in una variabile di "global.java".
Quando viene creato un nuovo file, viene lanciato il metodo "check" della classe "parser.java" che restituisce un booleanoo. Nel caso in cui il file

viene riconosciuto, il suo nome viene aggiunto alla lista degli esercizi presenti nel server. Anche la location della lista è memorizzata nel file "global.java" ed è scritta con un xml di questo tipo:

```
<esercizi>
    <esercizio>nome esercizio 1</esercizio>
    <esercizio>nome esercizio 2</esercizio>
    .
    .
    .
    <esercizio>nome esercizio N</esercizio>
</esercizi>
```

una volta che viene effettuata la relativa richiesta, viene inviato al client solo il nome degli esercizi, visto che l'indirizzo da cui scaricarli è noto.
Se invece il file non viene validato dal parser, la DirectoryWhatcher lo elimina.
Analogamente quando un file viene eliminato viene rimosso anche il suo nome dalla lista.
Nel caso di errori o di crash da parte del server, una volta che viene riavviato vengono controllati tutti i file presenti nella cartella, aggiungendoli nel caso in cui sono corretti eliminandoli altrimenti. Viene fatto un controllo analogo per la lista degli esercizi, in modo da eliminare dalla lista eventuali esercizi che sono stati rimossi.
Il procedimento di modifica del file non vene previsto. Infatti la funzione di upload non prevede di poter caricare due file con lo stesso nome. L'unico modo per modificare un file è aprirlo direttamente sulla macchina in cui è salvato. Questo procedimento comunque non può fallire. Infatti l'hash che viene inviato al client per poter verificare se un esercizio è stato modificato, viene calcolato al momento della richiesta.

3.5.2 richieste http:

Il secondo thread che viene creato si occupa di gestire la registrazione degli utenti.
Come ho già detto in precedenza, quando viene cliccato il pulsante registrazione dell'app, il server invia un email di conferma all'indirizzo fornito dall'utente contenente un numero casuale random (formato a sua volta da tre numeri compresi tra 10000000 e 100000000). Questo diventa l'id di registrazione indicato nel link che viene inviato. Per terminare la procedura quindi viene creato un thread che apre una connessione http. Cliccando sul link quindi viene ricercato nel database il nome dell'utente relativo alla richiesta. Non è detto che il nome sia presente nella relativa tabella (la "confMail", per approfondimenti si veda capitolo *3.6.2*). Infatti l'utente potrebbe già aver completato la richiesta o il tempo della richiesta può essere scaduto (il link è valido per 10 minuti). Nel caso in cui invece viene trovato, la registrazione termina e sulla pagina comparirà il messaggio per indicare che la procedura è andata a buon fine.

3.5.3 comunicazione col client

Una volta che il server accetta una richiesta, che avviene solo dopo l'autenticazione col certificato presente nel client, viene avviato un thread che si occupa di gestirla.
Il processo principale si metterà quindi nuovamente in ascolto attendendo nuove richieste.
Il superServer lascia la gestione delle varie richieste al guestServer, che si occupa di gestire le richieste di gestione credenziali o di controllare se un utente è già connesso: nel primo caso, se viene effettuata una richiesta di

login, aggiunge l'utente all'elenco di quelli connessi, nel secondo caso controlla la tabella degli utenti connessi e, se è presente, lascia la gestione della richiesta a un nuovo thread (personalServer) che si occuperà di gestirla.
La classe che si occupa di ricevere le informazioni del client, fare la relativa domanda al database e ritornare la relativa risposta è la "dbConnect".
I thread “server”, infatti, passano la stringa alla dbConnect che, una volta riconosciuto il tipo di richiesta che si sta facendo, lancia il relativo metodo che si occuperà di scomporre la stringa inviata dal client nelle varie informazioni, fare la richiesta al database mysql e trasformare la risposta in una stringa che rispetti il protocollo.
Passiamo quindi alla descrizione delle varie richieste e delle relative risposte.

-Richiesta 0, registrazione:

Il metodo “registration” si occupa di gestire la registrazione.
Viene controllato se l'utente è già presente nella tabella “utenti” del database; in caso di esito negativo, l'utente viene memorizzato nella tabella “confmail” e viene generato un numero casuale relativo all'utente (insieme a queste due informazioni viene anche memorizzato il timestamp unix della richiesta).
Infatti se non avviene la conferma di registrazione entro 10 minuti l'utente viene eliminato dalla tabella e deve quindi ripetere la procedura.
In caso di successo viene inviata un email contenente il link con le informazioni per la conferma della password. Nel caso in cui l'utente senza aver confermato la sua email tenta di registrarsi nuovamente, viene aggiornata la password con l'ultima inserita e viene inviata una nuova mail.

Risposte da parte del server:

Successo: mail-reg-ok

Errore: già-registrato

-Richiesta 1, login:

Una volta effettuata la registrazione è possibile eseguire il login.
Nella classe dbConnect troviamo il metodo login: questo controlla che l'utente sia presente nella tabella "utenti" del database, e in caso di esito positivo, chiama la funzione "putConnessi", che inserisce l'utente nella tabella degli utenti connessi e gli associa un numero casuale (chiave di sessione), che dopo l'autenticazione sarà usato per riconoscere l'utente (questo procedimento server per evitare di inviare ad ogni richiesta l'utente e la relativa password).
Un utente che non comunica col database per un tempo maggiore di 10 minuti viene automaticamente eliminato dalla tabella "connessi".
È da notare che ogni volta che un utente si identifica viene generato un valore casuale e viene inserito nella tabella, quindi è possibile usare più dispositivi contemporaneamente a nome di un utente senza che ci siano equivoci per quanto riguarda le richieste.

Risposte del server:

Successo: sessionKey

Errore: login-error

-Richiesta 2, recupero password:

Il metodo "recupero" serve per permettere all'utente di recuperare la password nel caso in cui l'abbia dimenticata.

Il metodo si limita ad inviare un email contenente la password all'indirizzo specificato.

Risposte del server:

Successo: mail-pass-ok

Errore: user-error

-Richiesta 3, conferma registrazione:

Questa richiesta viene fatta da uno dei due thread iniziali che sono stati creati, cioè quello che si occupa delle richieste HTTP.
Come già menzionato precedentemente, dopo la registrazione viene inviato un email contenente un link contenente un numero casuale a cui corrisponde l'username che ha fatto la richiesta,.
Cliccando sul link si effettua una richiesta GET al server.
Il thread del server risale all'ID dell'utente e lo sposta dalla tabella "confmail" a quella degli "utenti".
Il metodo utilizzato nella dbConnect è "confirmRegistration".

Una volta fatta la richiesta verrà stampata sulla pagina del browser la risposta del server.

Risposte del server:

Successo: registrazione-ok

Errore: registrazione-error

-Richiesta 4, richiesta lista esercizi:

Gli esercizi si trovano in una cartella del server. Possono essere inseriti tramite Web o manualmente mettendoli direttamente nella cartella.

Una volta che il programma server viene lanciato uno dei due thread iniziali (DirectoryWhatcher) controlla se tutti i file presenti nella cartella sono esercizi validi.
In caso positivo controlla se sono già stati inseriti nella lista degli esercizi, se non sono presenti vengono aggiunti, nel caso in cui il file non è un esercizio valido, questo viene eliminato.
Terminato il primo controllo il thread si mette in ascolto sulla cartella nel caso in cui venisse inserito o eliminato qualcosa. Ognuna di queste due azioni, infatti, verrà intercettata e andrà a modificare opportunamente la lista degli esercizi.
Questa lista è memorizzata nella stessa directory in cui si trova il codice ed è semplicemente un XML contenente tutti i nomi degli esercizi disponibili.
Per avere la lista degli esercizi quindi, basta parsare il file esercizi.xml. La classe che si occupa di questa attività è la "EsNameParser", che oltre a contenere il metodo "esName" che restituisce i nomi degli esercizi, contiene anche il metodo "check" che viene utilizzato dalla DirectoryWatcher per riconoscere se un esercizio caricato è valido oppure no.

Risposte del server:

Successo: esericizio1/esercizio2/esercizio3/.../esercizioN

Errore: null (anche quando non ci sono esercizi)

-Richiesta 5, informazione sugli utenti connessi:

Questa richiesta viene fatta dal guestServer e il metodo che si occupa della richiesta è "infoConnessi".
Questo metodo prende in input la sessionKey e controlla se un utente ha effettuato il login (naturalmente per le richieste di login, registrazione,

cambio password e recupero password non viene effettuata questa richiesta).

Risposte del server:

Successo: loggato

Errore: request-login

-Richiesta 6, settaggio del timer degli utenti connessi:

Come già specificato precedentemente,un utente che non effettua una comunicazione col server per 10 minuti viene rimosso dalla tabella connessi. Al termine di una comunicazione con il server, il personalServer chiama questo metodo per impostare l'ora dell'ultima connessione a quella attuale. È da notare che solo gli utenti che effettuano richieste che vengono gestite dal personalServer si trovano nella tabella connessi e quindi hanno bisogno di resettare il timer.

Risposte del server:

Successo: timer-settato

Errore: (non sono previsti casi di errore)

-Richiesta 7, esercizi con hash:

Questa richiesta viene effettuata per richiedere oltre ai nomi degli esercizi presenti nel database anche l'hash dell'esercizio. In questo modo nel momento in cui un esercizio viene aggiornato, il client si accorge del cambiamento e scarica la nuova versione del file

Risposte del server:

Successo: esercizio1/hash1/.../esercizioN/hashN

Errore: null (anche in caso di assenza di esercizi)

-Richiesta 8, accettazione di un esercizio:

Come già specificato nel paragrafo riguardante l'applicazione, ci sono due momenti distinti in cui avviene automaticamente la sincronizzazione: dopo aver effettuato il login e dopo aver terminato un esercizio.
Durante la sincronizzazione l'applicazione invia al server tutti gli esercizi che non sono stati ancora approvati.
Il metodo che si occupa di effettuare la convalida è exerciseAccept: questo prende in input una stringa contenente tutte le informazioni relative all'esercizio e tramite la sessionKey risale all'utente.
Per confermare l'esercizio viene inviato l'hash MD5 del contenuto dello stesso, viene quindi controllato con l'hash calcolato al momento della richiesta con quello dell'esercizio presente nel database.
In questo modo siamo sicuri che l'utente non ha modificato l'esercizio: senza questo controllo un utente malintenzionato potrebbe infatti cambiare l'esercizio rendendolo più semplice.
Solo una volta che l'MD5 viene convalidato vengono inseriti i dati relativi all'esercizio nel database, in modo che essi risultino visibili dall'insegnante.
Il client si preoccuperà di gestire il caso in cui l'esercizio non venga convalidato.

Risposte del server

Successo: es-ok

Errore: es-err

-Richiesta 9, esercizi completati:

Questa richiesta viene effettuata durante l'aggiornamento del client. il server restituisce tutti gli esercizi svolti dall'utente che ha effettuato la richiesta e che sono stati memorizzati nel database: questo permette all'applicazione di tenere sempre aggiornati tutti i dispositivi di un utente.

Risposta del server:

Successo: *nomeEs1/info1Es1/info2Es1/.../infoNEs1//*
nomeEs2/info1Es2/.../infoNEs2//...//
nomeEsM/info1EsM/.../infoNEsM

Errore: *null (anche nel caso in cui non ci sono esercizi)*

-Richiesta a0, cambio password:

Questa richiesta cambia semplicemente la password dell'utente specificato con la nuova password inserita.

Risposte del server:

Successo: cambiata

Errore: error

3.6 I Database:

Come già detto in precedenza, ci sono due database: LogicPlayerDB e exercise, il primo utilizzato dal server e dalla Web app il secondo dall'applicazione Android.

3.6.1 Exercise:

Ho implementato il primo database usando SQLite, libreria SQL implementata direttamente nei dispositivi Android.
Questo database è molto semplice e viene utilizzato per memorizzare le informazioni relative agli esercizi completati. Tutte le informazioni relative al database sono nella classe "personalTrackerContract". Per quanto riguarda invece i metodi di gestione, questi sono presenti nella classe "personalDBHelper".
Quindi per effettuare dei cambiamenti nel database basta modificare i campi nella "personalTrackerContract" e modificare i metodi nella "personalDBHelper".
Questa base di dati contiene una tabella chiamata "esercizi" strutturata nel modo seguente (tra parentesi il tipo di dato della colonna):

-esercizio (text): memorizza il nome dell'esercizio completato;
-md5 (text): hash dell'esercizio;
-utente (text): l'utente che ha completato l'esercizio;
-time(double): il momento in cui è stato completato l'esercizio;
-check(int): 1 se l'esercizio è stato validato dal server 0 altrimenti;
-voto(int): un valore numerico da 1 a 5 che indica la valutazione della prova.

La struttura del database ci permette di mantenere memorizzati nella stessa tabella tutti gli utenti di un dispositivo.

3.6.2 LogicPlayerDB:

Realizzato con mySQL, questo database lavora contemporaneamente sia con la Web app che col server. Per questo motivo ci sono due file che

contengono le informazioni relative a LogicPlayDB, il primo è in dbdata.php il secondo è in global.java nella sottoclasse “dbConstant”. Mentre per il server esiste una classe centrale (dbConnect) che si occupa di tutte le operazioni col database, nella Web app non esiste un unica classe. Ci sono invece diversi file PHP.
La base di dati contiene 5 tabelle:

-WebAdmin: contiene username e password degli amministratori della Web app;
-confMail: viene utilizzata per memorizzare tutte le email in attesa di conferma;
-connessi: contiene le informazioni degli utenti attualmente connessi o che si sono collegati negli ultimi 10 minuti;
-utenti: contiene username e password degli utenti;
-esercizi: contiene le informazioni relative agli esercizi effettuati dagli studenti.
Ogni tabella è costituita da una o più delle seguenti colonne:
-user (varchar): nome dell'utente;
-pass (varchar): password dell'utente;
-random (varchar): valore random (utilizzato per la verifica della mail);
-time (double): timestamp unix in millisecondi;
-tempKey (varchar): contiene un numero random associato a una connessione;
-id (int): distingue gli esercizi;
-esercizio (varchar): contiene il nome di un esercizio;
-voto (int): contiene il voto di un utente rispetto a un esercizio.

Capitolo 4:
Conclusioni e sviluppi futuri

L'applicazione sviluppata risulta complessivamente aderente agli obiettivi che ci eravamo posti in fase di progettazione.
Il backend dell'app implementa in maniera complessivamente soddisfacente le feature minime richieste per un'adeguata esperienza utente.
Anche il server è complessivamente abbastanza affidabile e non ha mostrato, durante l'attività di testing, problemi o bug tali da rendere inservibile l'applicazione Android.
Alcune funzionalità, per ragioni di tempo, risultano tuttavia ancora imprecise e migliorabili:
- I tasti azione dell'interfaccia di Android non sono configurati correttamente, e spesso capita di tornare allo svolgimento di un esercizio già terminato attraverso la pressione accidentale del tasto *Back*
- L'invio automatico di e-mail per la conferma degli account è stato gestito attraverso un account Gmail di Google che è stato recentemente inattivato a causa della mole eccessiva di posta in uscita prodotta automaticamente
- La pressione molto rapida e continua del tasto aggiorna (e più in generale la richiesta quasi simultanea di molte funzioni diverse del server) ha a volte mandato in crash l'intero sistema a causa di un'implementazione ancora non perfetta delle varie routine di gestione delle eccezioni.
- Gli esercizi non vengono mai eliminati dal database locale del dispositivo, tranne quando vengono completati. Questo potrebbe comportare un sovraccarico di informazioni vecchie e potenzialmente

inutili, oltre a risultare problematico nell'eventualità in cui un esercizio venga rimosso dalla memoria del dispositivo.

Per quanto riguarda gli sviluppi futuri, per prima cosa si potrebbe pensare al riconoscimento della firma dell'applicazione da parte del server. Infatti al momento il server risponde a qualunque richiesta effettuata da un client che abbia il suo stesso certificato (in modo da creare la comunicazione sicura). Non essendoci altre protezioni, un utente malevolo potrebbe ricompilare l'applicazione modificando i vari controlli che vengono fatti durante la valutazione di un esercizio o potrebbero essere effettuate al server richieste differenti da quelle previste, mandando in crash l'intero sistema.
Il linguaggio utilizzato per scrivere gli esercizi potrebbe essere reso compatibile con quello utilizzato da TPTP, utilizzando un linguaggio di trasformazione come XSLT. In questo modo è possibile al momento dell'inserimento controllare se un esercizio è risolvibile oppure no. Si potrebbe inoltre, basandosi su questi risultati, creare un meccanismo di valutazione automatizzato, che non obblighi quindi l'insegnante a comunicare in anticipo il punteggio ottimale di un esercizio.
Sempre per la parte riguardante il punteggio, potrebbero essere implementati metodi più fiscali per far in modo di controllare utilizzi malevoli da parte degli utenti. Per il momento infatti nulla toglie all'utente la possibilità di svolgere gli esercizi su un dispositivo fino a trovare la soluzione ottima, non sincronizzare mai quel dispositivo e risolvere lo stesso esercizio su un secondo device permettendogli così di prendere il massimo del punteggio.
Si potrebbe inoltre migliorare la Web-app aggiungendo informazioni utili all'insegnante, come statistiche o grafici. Inoltre sarebbe utile migliorare la grafica in modo da permettere ricerche più accurate o aggiungere maggiori meccanismi per l'amministrazione.

Riguardo la scalabilità dell'applicazione a logiche diverse o a formalismi diversi, ciò comporterebbe sforzi minimi dal punto di vista del backend: sia il server che l'infrastruttura Web per la gestione della classe sono infatti del tutto indipendenti dalla tipologia di esercizi, e sarebbe sufficiente cambiare le metriche di valutazione dell'app e il formato XML per la codifica dell'esercizio.

Bibliografia

[1] “Java API - Oracle Documentation”, Java API - Oracle Documentation

[2] “Android API Reference”, http://developer.android.com/reference/packages.html

[3] “Android API Guide”, http://developer.android.com/guide/index.html

[4] “Stack Overflow”, http://stackoverflow.com

[5] “JDOM”, http://www.jdom.org

[6] Bedogni L., Di Felice M., “Android Laboratory: Program and Resources”, http://www.cs.unibo.it/projects/android/2014/index.html#resources

[7] “Javax.mail”, http://www.oracle.com/technetwork/java/index-138643.html

[8] “Download Connector/J”, http://dev.mysql.com/downloads/connector/j/

Appendice: note sul codice prodotto

In questa appendice verranno elencati e descritti i vari strumenti per lo sviluppo delle relative parti del codice e ne verranno analizzate le dimensioni e alcune caratteristiche tecniche.
Riguardo all'intero progetto, ho stimato una lunghezza di circa 3500 righe di codice per un lavoro complessivo di 180 ore persona.
Di queste ore la maggior parte è stata spesa nella realizzazione della connessione sicura tramite SSL (che ha richiesto uno studio approfondito del sistema dei certificati di Android e diverse sperimentazioni sul modo migliore per collegare un dispositivo Android ad un server Java) e nella realizzazione del parser per la conversione di stringhe in Formule.

Applicazione Android:
L'applicazione è stata sviluppata con AndroidStudio 0.83, lo strumento fornito da Google (per ora ancora in versione beta) dedicato alla programmazione per app Android.
L'IDE si è rivelato affidabile e ha soddisfatto pienamente le aspettative e le necessità progettuali.
L'applicazione è stata sviluppata per Android 4.4 utilizzando l'API 20 e testata su un tablet Nexus 7 2012 con Android Kit Kat 4.4 e un cellulare HTC Desire 500 con Android Jelly Bean 4.1.2. L'unica libreria aggiuntiva utilizzata è JDOM2, la libreria che viene utilizzata dal parser contenente i vari comandi per poter utilizzare XPATH.

Server:
Il codice del server è stato scritto in Java versione 1.8 utilizzando IntellijIdea 14 per il suo sviluppo.
Sono state inoltre usate le seguenti librerie:

-JDOM2, come per il parser in Android anche quello del server utilizza questa libreria per leggere o modificare i file XML.
-JAVAX MAIL, libreria utilizzata per inviare le email per confermare l'email di registrazione degli utenti.
-MySQL CONNECTOR, utilizzata per far interagire il server con il database.
Per poter convertire i certificati JKS in BKS è stato utilizzato il programma Portacle rilasciato dalla Bouncy Castle. Il software permette di trasformare i certificati JKS (riconosciuto da Java) in BKS (riconosciuto da Android).

-Web App:

Per quanto riguarda l'applicazione Web, non sono stati utilizzati particolari strumenti di supporto alla programmazione.
Per implementare la Web app è stata utilizzata la piattaforma Apache 2, ho utilizzato PHP 5 per scrivere gli script server client, utilizzando le varie librerie implementate per comunicare con i database SQL e per leggere i file XML.